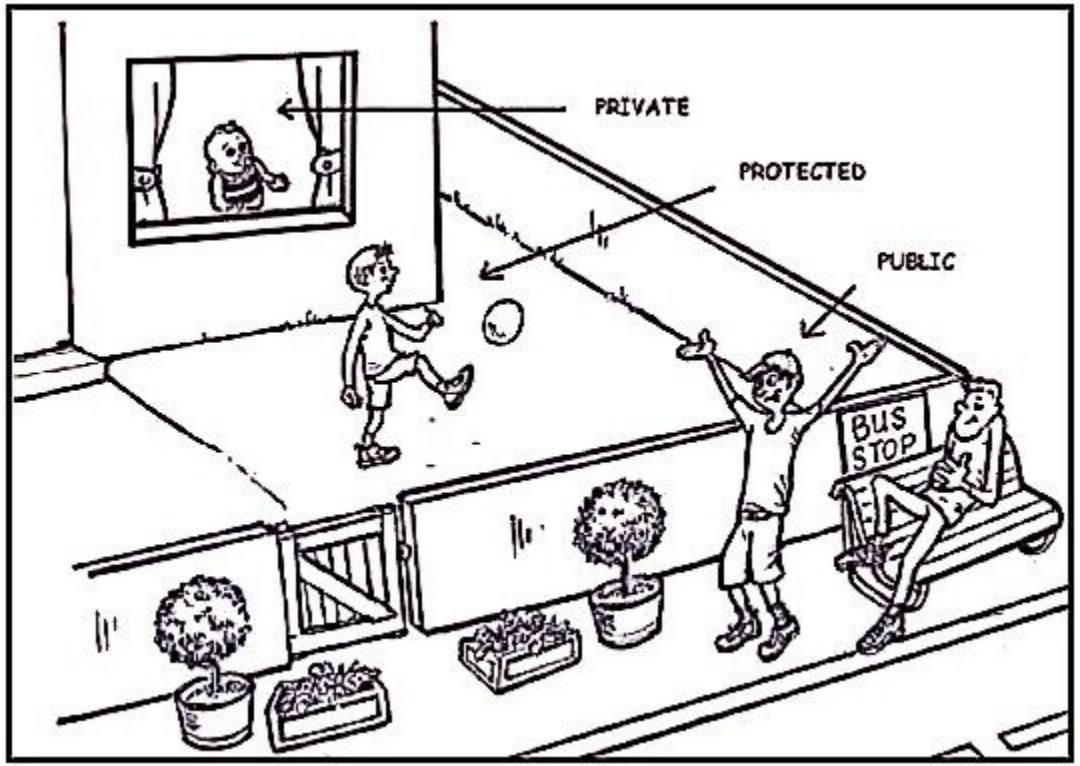
Область видимости переменных

Область видимости переменных

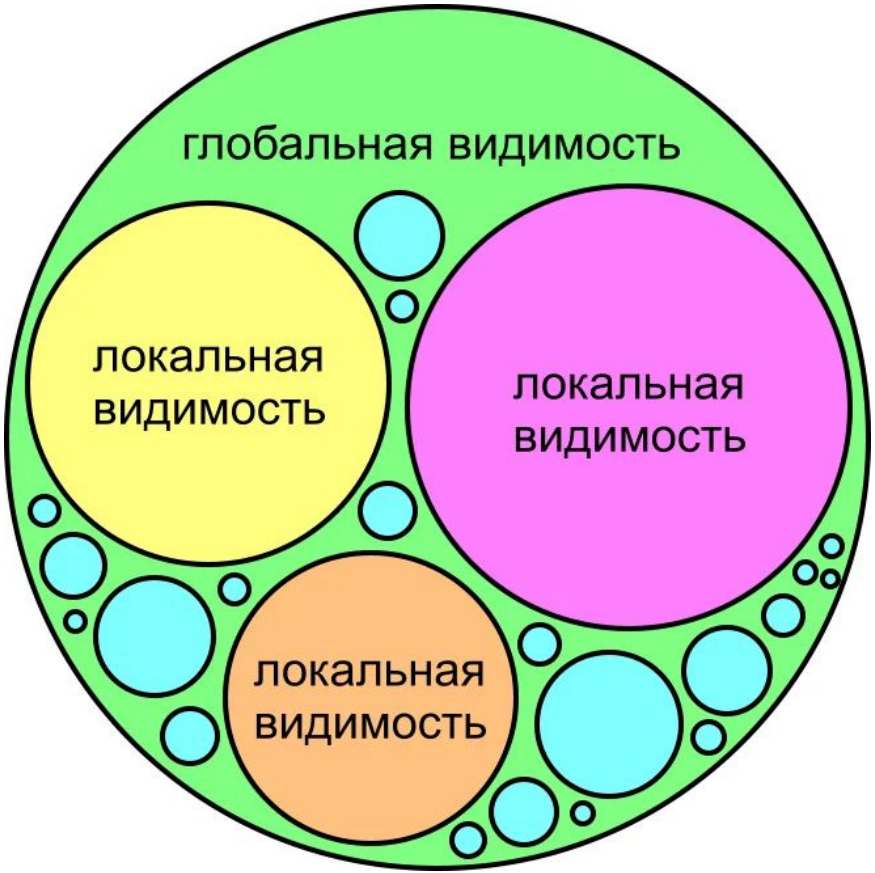


Область видимости, или контекст переменной — это часть кода, в пределах которого доступна данная переменная.

Существуют различные контексты:

- Контекст класса. Переменные, определенные на уровне класса, доступны в любом методе этого класса
- Контекст метода. Переменные, определенные на уровне метода, являются локальными и доступны только в рамках данного метода. В других методах они недоступны
- Контекст блока кода. Переменные, определенные на уровне блока кода, также являются локальными и доступны только в рамках данного блока. Вне своего блока кода они не доступны.

Глобальные и локальные переменные



Глобальные переменные видны во всей программе. "Видны" – значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

К глобальной переменной можно обратиться из локальной области видимости. К локальной переменной нельзя обратиться из глобальной области видимости, потому что локальная переменная существует только в момент выполнения тела функции. При выходе из нее, локальные переменные исчезают. Компьютерная память, которая под них отводилась, освобождается. Когда функция будет снова вызвана, локальные переменные будут созданы заново.

Все члены класса - поля, методы, свойства - все они имеют **модификаторы доступа**. Модификаторы доступа позволяют задать допустимую область видимости для членов класса. То есть модификаторы доступа определяют контекст, в котором можно употреблять данную переменную или метод.

В C# применяются следующие модификаторы доступа:

- **public**: публичный, общедоступный класс или член класса. Такой член класса доступен из любого места в коде, а также из других программ и сборок.
- **private**: закрытый класс или член класса. Представляет полную противоположность модификатору public. Такой закрытый класс или член класса доступен только из кода в том же классе или контексте.
- **protected**: такой член класса доступен из любого места в текущем классе или в производных классах. При этом производные классы могут располагаться в других сборках.
- **internal**: класс и члены класса с подобным модификатором доступны из любого места кода в той же сборке, однако он недоступен для других программ и сборок (как в случае с модификатором public).
- **protected internal**: совмещает функционал двух модификаторов. Классы и члены класса с таким модификатором доступны из текущей сборки и из производных классов.
- **private protected**: такой член класса доступен из любого места в текущем классе или в производных классах, которые определены в той же сборке.

Переменные, объявляемые как static, по существу, являются глобальными. Когда же объекты объявляются в своем классе, то копия переменной типа static не создается. Вместо этого все экземпляры класса совместно пользуются одной и той же переменной типа static. Такая переменная инициализируется перед ее применением в классе.


Типы данных
Типы значений
Ссылочные типы
Логический тип
bool
Символы
char
Десятичный тип
decimal
Типы с плавающей запятой
float
double
Целочислен-ные типы
byte
sbyte
short
ushort
int
uint
long
ulong
Объекты
object
Строки
string

SerializeField

Сериализация - это по сути любой процесс "обертки данных". Например, тип Vector3 может быть сохранен как три числа x, y, z и затем восстановлен из этих чисел обратно в Vector3. В грубом виде - это и есть та самая сериализация.

В Unity, когда ваш код комплируется после внесенных изменений (в народе это называют ребилдом), либо когда вы входите/выходите из игрового режима - данные, введенные ранее, остаются, но вместе с тем добавляются новые поля. Фактически, происходит программирование "налету" и этому Unity обязан своей сериализации.

SerializeField

В проекте по умолчанию сериализуются только поля с доступом public. Если поле сериализуется - оно появится в стандартном инспекторе и его свойства можно будет настраивать.